BBC RD 1987/19

BBC

Research Department Report

November 1987

DESIGNING VARIABLE-LENGTH CODES WITH GOOD ERROR-RECOVERY PROPERTIES

M.J. Knee, M.A.

Research Department, Engineering Division
THE BRITISH BROADCASTING CORPORATION

BBC RD 1987/19

UDC 621.397: 621.3.088

DESIGNING VARIABLE-LENGTH CODES WITH GOOD ERROR-RECOVERY PROPERTIES

M.J.Knee, M.A.

Summary

An algorithm for calculating the expected resynchronization time of a variable-length code following a transmission error is described. The results of calculations based on examples are used, together with insights gained from a simplification of the algorithm, to determine the effect on the expected resynchronization time of certain easily recognizable design features of codes. The Report concludes that, if a code is designed carefully according to a few simple principles, its error-recovery properties may be significantly better than those resulting from a straightforward application of, for example, the Huffman algorithm. Such improvements are particularly important in the design of codes for digital television bit-rate reduction systems.

Issued under the Authority of

M.E.B. Roffey

Head of Research Department

**Research Department, Engineering Division,
BRITISH BROADCASTING CORPORATION**

(EL-190)

November 1987

DESIGNING VARIABLE-LENGTH CODES WITH GOOD ERROR-RECOVERY PROPERTIES

M.J.Knee, M.A.

DESIGNING VARIABLE-LENGTH CODES WITH GOOD ERROR-RECOVERY PROPERTIES

M.J.Knee, M.A.

1. INTRODUCTION

The BBC has been investigating several proposed picture coding methods[1] for transmission of a contribution-quality digital *YUV* picture signal, sampled according to CCIR Recommendation 601, at bit rates up to 53 Mbit/s. Most of the methods studied use a combination of differential pulse code modulation (DPCM) and variable-length coding of the resulting quantized prediction error signal to achieve the necessary reduction in bit rate[2]. The use of variable-length coding makes a significant contribution to this reduction, but it also gives rise to problems of synchronization loss following an error in the transmitted bit-stream. This is because if a codeword is received in error, it may also be interpreted as a codeword of a different length. In general, the decoder will eventually resynchronize with the coder and correct interpretation of the transmitted bit-stream will be resumed. Clearly, it is desirable to limit the effect of synchronization loss as much as possible.

The Report discusses methods of determining one important resynchronization parameter, the expected resynchronization time, of variable-length codes and, by using simple examples, illustrates how it is affected by certain easily recognizable features of the codes. This leads to the establishment of some principles which can be applied when attempting to design practical codes with good error recovery properties.

The examples in this Report are related to DPCM coding, but the principles can be applied to any other bit-rate-reduction algorithm such as transform coding or adaptive subsampling.

2. DEFINITIONS AND ASSUMPTIONS

2.1 Entropy and bit rate

We are concerned with choosing binary codewords for a set of n possible source symbols A_i, $i = 1,...,N$. For example, the A_i might be the output levels of a nonlinear DPCM quantizing law. We assume that the A_i occur independently with probability P_i. The *entropy*, H, of the source is then given by

$$H = -\sum_{i=1}^{N} P_i \log_2 P_i$$

and is a lower bound, in bits per symbol, of the average bit rate attainable using a variable-length code. As an example, if $N = 7$ and $\{A_i\} = \{0, +1, -1, +2, -2, +3, -3\}$ is a set of DPCM output levels, we might have $\{P_i\} = \{0.3, 0.2, 0.2, 0.1, 0.1, 0.05, 0.05\}$. The entropy of this source is then 2.55 bits per symbol.

To design a code for the source, we assign a codeword C_i, of length L_i bits, to each symbol. Since we are concerned with variable-length codes, the L_i are not all equal. The average bit rate of the code is given by

$$B = \sum_{i=1}^{N} P_i L_i$$

It is essential that the decoder can correctly determine the boundaries between transmitted codewords. This is most conveniently ensured by making the set of codewords obey the *prefix condition*, whereby no codeword is the prefix of any other. This paper will only be concerned with such *prefix codes*. However, it should be stressed that the prefix condition is not necessary and that other codes may have better error recovery properties.

Huffman[3] presents the classical algorithm for designing an *optimal* code, i.e. one whose average bit rate approaches the entropy as nearly as possible. This algorithm, which produces a prefix code, seems to be the one most commonly used in proposed bit-rate reduction systems. The algorithm is reproduced in Appendix 1.

Applying Huffman's algorithm to the above example source, we obtain Code 1 given in Table 1. The set $\{L_i\}$ is thus {2, 2, 3, 3, 3, 4, 4} and the average bit rate is 2.6 bits per symbol.

Note that Code 1, like all the other codes considered in this Report, is an *exhaustive* code. This means that any long sequence of bits can eventually be interpreted as a sequence of codewords. A code consisting of the first six codewords of Code 1 would not be exhaustive, since a sequence of bits beginning with 1101 could never be interpreted as a sequence of codewords. A non-exhaustive code can be turned into an exhaustive code by the addition of the necessary codewords with their associated probabilities P_i set to 0.

Table 1
Examples of variable-length codes

Symbol (C_i)	Probability (P_i)	Code 1	Code 2	Code 3	Code 4	Code 5	Code 6	Code 7
0	0.3	00	00	00	00	00	00	00
+1	0.2	10	10	11	01	11	01	01
−1	0.2	010	010	010	100	011	100	10
+2	0.1	011	011	011	110	010	101	1100
−2	0.1	111	110	101	111	100	111	1101
+3	0.05	1100	1110	1000	1010	1010	1100	1110
−3	0.05	1101	1111	1001	1011	1011	1101	1111

2.2 Describing codes

The set of codeword lengths is often written as a list of indices giving the number of codewords having each possible length. For example, Code 1 is known as a (0232) code, indicating that it has no codewords of length 1, two of length 2, three of length 3 and two of length 4.

A useful pictorial representation of a prefix code is the *tree diagram*. For example, Fig. 1(a) is a tree diagram of Code 1. The *root* of the tree represents the start of every codeword and the *terminal nodes* represent the ends of the codewords. Each codeword is represented by a path along the branches of the tree from the root to a terminal node, an upward branch representing a **0** and a downward branch a **1**. The fact that the code obeys the prefix condition ensures that each codeword leads to a unique terminal node which is not 'on the way' to any other terminal node. The process of decoding a message can be thought of as travelling from left to right, taking upward or downward branches according to the received bit-stream, and returning to the root whenever a terminal node is reached.

2.3 Errors and their effects

We shall be concerned only with errors of the bit-reversal kind; that is, a transmitted **0** is received as a **1** or vice versa. Errors consisting of a bit loss or a bit gain are not expected to be encountered in a transmission system in which the (smoothed-out) transmitted bit rate is fixed. We assume that the probability of an error is independent both of time and of the state of the signal.

When an error occurs, the *resynchronization time*, *s*, is defined as the number of consecutive source symbols incorrectly decoded, including the one in which the error itself occurs. We assume that errors occur infrequently enough for the effect of a second error occurring before resynchronization to be ignored.

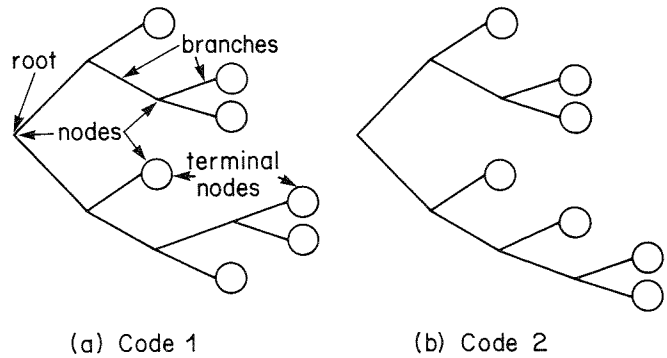


Fig. 1 - Tree diagrams of Codes 1 and 2.

An important measure of the error recovery properties of a code is *Es*, the expected value of *s*.

The methods used in this Report to calculate the value of *Es* for a particular code involve the analysis of the code in terms of the possible *states* in which the decoder can be left after a (possibly corrupted) codeword has been received and after as many whole codewords as possible have been removed. In the absence of errors, each codeword is successfully decoded and the decoder is always left in the *synchronized state*. In the code's tree diagram, this state corresponds to the root and the terminal nodes (considered collectively). For convenience, the synchronized state in which the decoder begins just before it receives a corrupted codeword is called the *initial state*, *I*, while the state to which the decoder eventually resynchronizes is the *resynchronized state*, *S*.

Following a corrupted codeword, the decoder might be left in one of several *error states*, T_j, corresponding to the intermediate nodes in the tree diagram. For example, if codeword 111 in Code 1 is received as 110 because of an error, the decoder is said to be in error state 110 because it can make no sense of this sequence of bits until it receives another codeword. If the next codeword is 10, the decoder will then contain the sequence 11010. It will recognize

1101 as a codeword and will thus be left in error state 0. If it then receives 10, it will contain the sequence 010 which is a codeword, so it will be left in state S and resynchronization will have been achieved. It can be seen from the above that, for an exhaustive code, the error states are all sequences that are prefixes of codewords. The error states of Code 1 are thus 0, 1, 01, 11 and 110.

A sequence of bits is a *synchronizing sequence* if, when received by the decoder, synchronization is guaranteed whatever state the decoder is in to begin with. This means that the sequence of bits forms a sequence of codewords both on its own and when preceded by any error state. For example, the sequence 0010 is a synchronizing sequence for Code 1. If a synchronizing sequence is a single codeword it is known as a *synchronizing codeword.* It should be noted that in this Report we are only concerned with states that can actually be attained by the decoder following a bit-reversal error. Thus, for example, in a code whose codewords are all of even length, we do not consider states consisting of an odd number of bits.

3. PREVIOUS WORK

A number of workers have tackled the problem of error recovery in variable-length codes. The following is a very brief survey of some of the literature.

Rudner[4] reports that there is thought to be no fast algorithm for generating an optimal code that also minimizes the expected resynchronization time. However, he argues that the length of the shortest synchronizing sequence is a good measure of the error recovery properties of a code, and he presents an algorithm for generating a code, with a given set of codeword lengths, that minimizes this parameter.

Neumann[5] characterizes a class of slightly sub-optimal codes with good error recovery properties. In another paper[6], he considers a very interesting class of slightly sub-optimal conditional entropy codes, designed specifically for picture coding applications, with good error recovery properties. Lack of time has prevented a proper evaluation of this work. However, conditional entropy coding has been proposed[7] as a picture coding method and so this might be an area worthy of further study.

Hatcher[8] generalizes the codes of Neumann to include error-correcting capabilities. However, the codes he presents are extremely inefficient.

Ferguson and Rabinowitz[9] characterize codes whose word lengths are the same as those of the Huffman code and have at least one synchronizing codeword. Such codes are sometimes known as *self-synchronizing* codes. It will become apparent in this Report that such a feature is neither necessary nor sufficient to guarantee good error recovery properties.

A recent paper by Maxted and Robinson[10] provides much of the background for this Report. They describe a method of calculating the expected resynchronization time of a code and discuss several examples. The following section illustrates the use of this method on some examples and expands slightly on the published work.

4. CALCULATING THE EXPECTED RESYNCHRONIZATION TIME

The technique described by Maxted and Robinson is to calculate a *transition probability* for each ordered pair of decoder states. Recall that the possible states are the error states T_j, the initial state I and the resynchronized state, S. If A and B are two states, the transition probability $P(B|A)$ is the probability that, given decoder state A, it will be left in state B after it has received the next codeword. When the transition probabilities have been calculated, a *signal flow graph* can be drawn linking the various states together with weighted, directed links. The weights are polynomials in an arbitrary variable, z (say). Initially, each weight is equal to pz, where p is the transition probability corresponding to the link. Graph reduction transformations are then applied to the signal flow graph to eliminate all the error states and to end up with a polynomial in z linking states I and S. Fig. 2 gives an example of how a state is eliminated; this example embodies all the standard transformations required.

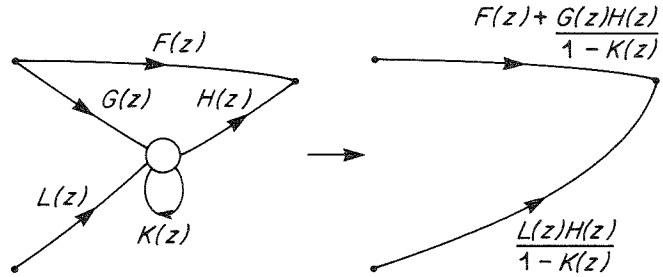


Fig. 2 - Example of a graph-reduction transformation.

The transformations ensure that, at each stage, the coefficient of z^m in the binomial expansion of the polynomial linking states A and B is the probability that, given state A, state B will be reached in a time corresponding to exactly m source symbols. Thus, when all the error states have been eliminated, the coefficient of z^m in the binomial expansion of the polynomial linking the final two states, I and S, is the

probability that the resynchronization time, s, is exactly m symbols. The expected resynchronization time, Es, can therefore be calculated by differentiating the polynomial and evaluating it at $z = 1$.

Table 2 illustrates how the transition probabilities can be calculated for Code 1. The transition probabilities $P(T_j|I)$ and $P(S|I)$ are calculated by looking at the effect of an error in each bit of each codeword. For example, the effect of an error in the first bit of the codeword 010 is to produce error state T_5 (i.e. 110), while an error in the fourth bit of 1100 leads to state S, since 1101 is also a codeword. In the diagram, the superscripts are the states resulting from each possible error. The contribution to the transition probability from an error in one bit of codeword C_i is P_i/B, where B is the average bit rate of the code ($B = 2.6$ in this example). The other transition probabilities $P(T_j|T_k)$ and $P(S|T_k)$ are calculated by looking at the effect of each codeword appended to each error state in turn. For example, error state T_5 followed by 011 gives the sequence 110011, from which 1100 would be decoded, leaving 11 which is error state T_4.

Fig. 3 shows the signal flow graph for Code 1. Unfortunately, the process of applying the graph reduction transforms is too cumbersome to illustrate, even for a code as small as this. However, a computer program has been written to calculate the value of Es using this method. The algorithm used is summarized in Appendix 2.

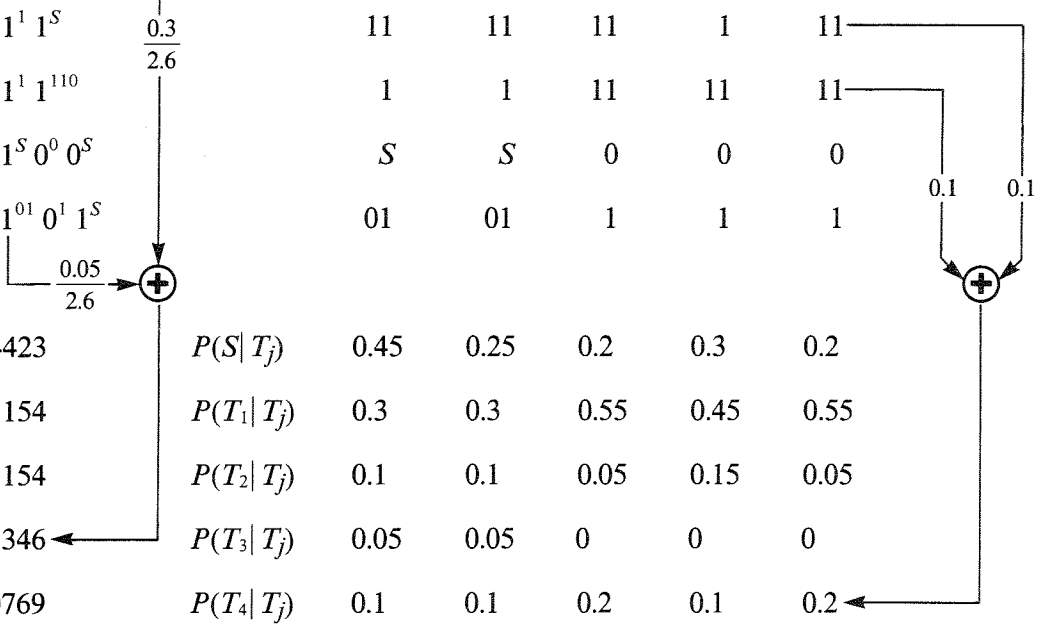
Maxted and Robinson illustrate (without proof) that a close estimate of Es can be obtained by ignoring any distinction between the error states and treating them as one error state T, giving the model shown in Fig. 4, where P_∞ is the steady-state transition probability from the error state T to the resynchronized state S. P_∞ can be calculated from the transition probabilities already derived, using the formula

$$P_\infty = \lim_{m\to\infty} \frac{((\mathbf{A}^m\mathbf{I}).\mathbf{S})}{|\mathbf{A}^m\mathbf{I}|}$$

where $\mathbf{A}$ is the matrix with elements $A_{jk} = P(T_j|T_k)$, $\mathbf{S}$ is the vector with elements $S_k = P(S|T_k)$ and $\mathbf{I}$ is the vector with elements $I_j = P(T_j|I)$. The process of repeatedly multiplying by $\mathbf{A}$ converges quite rapidly.

Table 2
Obtaining the transition probabilities for Code 1

P_i	C_i	Error states T_j:	0 (T_1)	1 (T_2)	01 (T_3)	11 (T_4)	110 (T_5)
0.3	$0^S\,0^{01}$		0	0	0	S	0
0.2	$1^S\,0^{11}$		S	110	0	0	0
0.2	$0^{110}\,1^0\,0^S$		S	S	S	0	S
0.1	$0^S\,1^1\,1^S$		11	11	11	1	11
0.1	$1^S\,1^1\,1^{110}$		1	1	11	11	11
0.05	$1^0\,1^S\,0^0\,0^S$		S	S	0	0	0
0.05	$1^1\,1^{01}\,0^1\,1^S$		01	01	1	1	1
$P(S\|I)$	0.4423	$P(S\|T_j)$	0.45	0.25	0.2	0.3	0.2
$P(T_1\|I)$	0.1154	$P(T_1\|T_j)$	0.3	0.3	0.55	0.45	0.55
$P(T_2\|I)$	0.1154	$P(T_2\|T_j)$	0.1	0.1	0.05	0.15	0.05
$P(T_3\|I)$	0.1346	$P(T_3\|T_j)$	0.05	0.05	0	0	0
$P(T_4\|I)$	0.0769	$P(T_4\|T_j)$	0.1	0.1	0.2	0.1	0.2
$P(T_5\|I)$	0.1154	$P(T_5\|T_j)$	0	0.2	0	0	0



Then, using the graph reduction transformation to eliminate the error state T, the polynomial linking I to S becomes

$$P(S|I)z + \frac{(1 - P(S|I))P_\infty z}{1 - (1 - P_\infty)z}$$

The expected resynchronization time is therefore given by

$$Es_\infty = 1 + \frac{1 - P(S|I)}{P_\infty}$$

If we make the further assumption that P_∞ is approximated by the probability of resynchronization following the first occurrence of the error state, the calculation of Es is simplified further. This approximate value, P_0, is given by

$$P_0 = \frac{\mathbf{I.S}}{1 - P(S|I)}$$

since **I.S** is the probability of reaching an error state after one symbol and resynchronizing after the next and $1 - P(S|I)$ is the probability of reaching an error state after one symbol. Thus, there is no need to calculate the $P(T_j|T_k)$, the estimate of Es being given by

$$Es_0 = 1 + \frac{(1 - P(S|I))^2}{\sum_{j=1}^{N} P(T_j|I)P(S|T_j)}$$

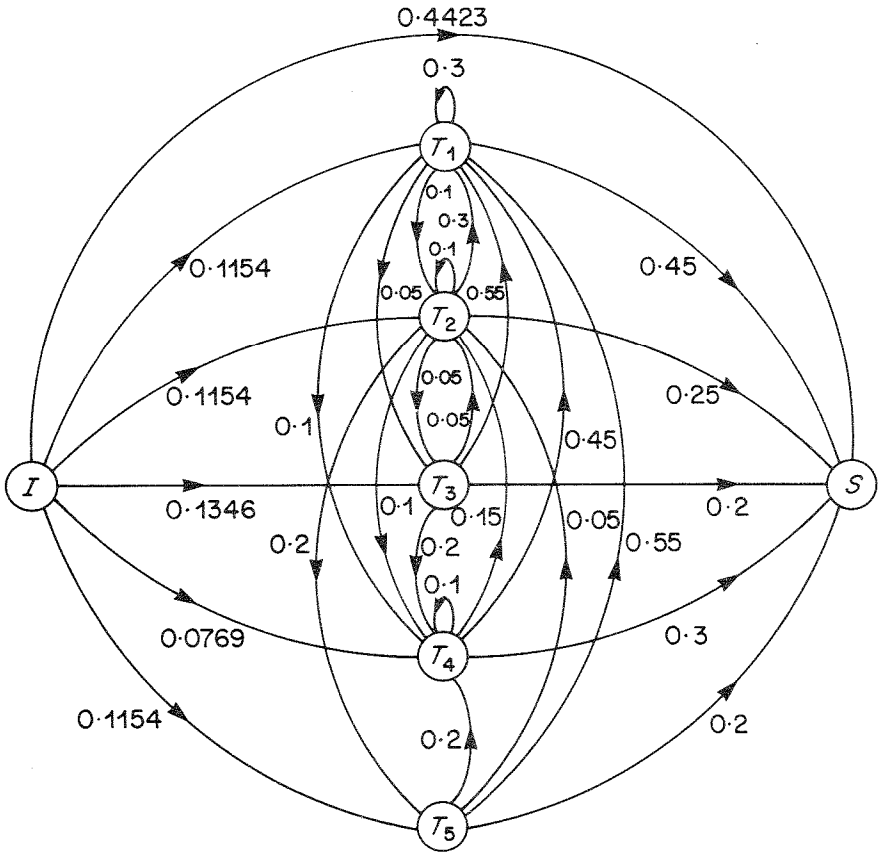


Fig. 3 - Signal flow graph for Code 1. (For clarity, the indeterminate, z, has been omitted.)

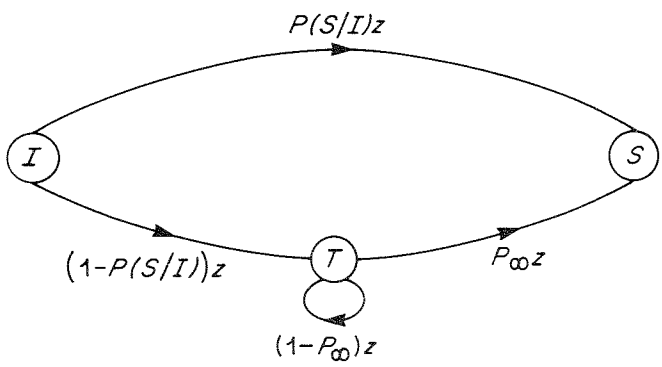


Fig. 4 - Signal flow graph for the simplified model.

No claims are offered for the validity of this second approximation but, in the examples following, both formulae will be used so that a comparison can be made.

5. COMPARISON BETWEEN CODES

In this section, we compare the performance of several codes using the exact method of calculating Es and the simplified and highly simplified estimates of Es derived above. To begin with, we look at six codes with the same codeword lengths, and therefore the same average bit rate, as Code 1.

5.1 Comparison between six (0232) codes

The codes used are presented in Table 1; their tree diagrams are in Figs. 1 and 5. Code 1 has been introduced already. It is the result of applying Huffman's algorithm (Appendix 1) to the probabilities given in Section 2.1. It should be noted that there are certain arbitrary choices in the implementation of Huffman's algorithm. The code used is meant to be an example of what would be obtained if no particular attempt were made to optimize those choices.

Code 2 (Fig. 1(b)) is the result of applying Rudner's algorithm[4] to the same set of codeword lengths. Recall that this algorithm minimizes the length of the shortest synchronizing sequence. Indeed, it does better than Huffman's algorithm in this respect; 010 is a synchronizing sequence, whereas Code 1 has no synchronizing sequence shorter than four bits long.

Codes 3 to 6 (Fig. 5) have been designed to illustrate the effects of two loosely defined conditions, both of which are thought to improve the error recovery properties of codes. The first condition, which we call L, is that as many codewords as possible are such that an error produces another codeword of the same length. This condition tends to increase the value of $P(S|I)$ and so at least the highly simplified estimate of Es, namely Es_0, will tend to be reduced as desired. The second condition, which we call X, is that as many codewords as possible are suffixes of other codewords. This condition tends to increase the probabilities $P(S|T_j)$, so once again Es_0 will tend to be reduced.

Code 3, which we call $\overline{L}\overline{X}$, has been designed heuristically to disobey both conditions as far as possible. Code 4 ($L\overline{X}$) has the L condition but disobeys X, Code 5 ($\overline{L}X$) has X but disobeys L, while Code 6 (LX) obeys both conditions as far as possible.

Table 3 gives the results of calculating and estimating Es for each of the six codes.

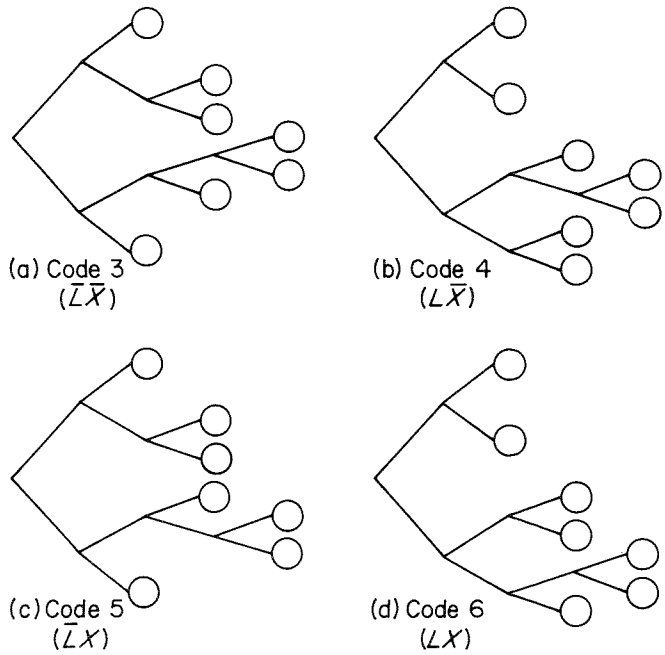


Fig. 5 - Tree diagrams of Codes 3 to 6.

In these results, both Es_∞ and Es_0 turn out to be reasonably good estimates of Es. The results also show that, as expected, $P(S|I)$ depends strongly on condition L, while P_∞ depends strongly on condition X. The results also show that condition X is the more important. Rudner's algorithm gives a nearly optimal result, but it should be pointed out that, in general, it ensures the existence of only one minimum-length synchronizing sequence. In these examples, Code 2 does have only one synchronizing sequence of length 3 (010), while Code 6 has two such sequences (100 and 101).

The most important outcome of these calculations is that the value of Es for a badly designed code is three times as large as its value for a well-designed code, and that neither Huffman's algorithm nor Rudner's necessarily gives the best results.

Table 3
Results of estimating Es for Codes 1 to 6

Code		Es	$P(S\|I)$	P_∞	P_0	Es_∞	Es_0
1	Huffman	2.65	0.4423	0.2695	0.2759	3.07	3.02
2	Rudner	2.19	0.4808	0.4500	0.4352	2.15	2.19
3	$\overline{L}\overline{X}$	5.36	0.1923	0.1871	0.1798	5.32	5.49
4	$L\overline{X}$	3.90	0.4231	0.2065	0.1667	3.79	4.46
5	$\overline{L}X$	2.61	0.1923	0.4500	0.5107	2.80	2.58
6	LX	1.87	0.4615	0.7000	0.5786	1.77	1.93

5.2 Codes with other sets of codeword lengths

The purpose of this section is to illustrate that the initial choice of the codeword lengths L_i may be just as important as the selection of codewords with a given set of lengths. As an example, we use the same probabilities as before, but consider a (0304) code, which happens to lead to the same bit rate (2.6 bits per symbol) as the (0232) codes considered above. This new set of lengths leads to an obvious code design that satisfies both the L and the X conditions very strongly. This is Code 7, and is given in Table 1 and Fig. 6. Estimates of Es were made for this code; the results are given in Table 4.

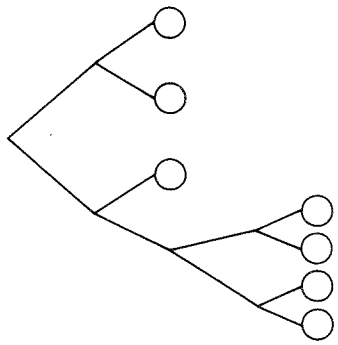


Fig. 6 - Tree diagram of Code 7.

Table 4
Comparison of the best (0232) and (0304) codes

Code	Es	$P(S\|I)$	P_∞	P_0	Es_∞	Es_0
6	1.87	0.4615	0.7000	0.5786	1.77	1.93
7	1.20	0.8077	0.9500	0.9500	1.20	1.20

Note that, since only one error state (11) is ever reached with Code 7, the estimates of P_∞ and P_0 are the same and, moreover, both of the simplified methods of estimating Es give an exact result for this code.

This example illustrates that Huffman's algorithm may not produce the best set of codeword lengths for error recovery purposes. In general, alternative sets of lengths for a code will lead to a slightly sub-optimal bit rate but, since the statistics of pictures are far from stationary, the probability distribution used to derive the lengths is somewhat arbitrary in any case.

It should be noted that, if bit-loss or bit-gain errors were possible, Code 7 would be a disastrous choice because all the codewords are of even length. Following such an error, synchronization would be lost until another such error occurred, unless an explicit resetting strategy were being used.

5.3 Examples from a DPCM system

The final examples in this paper are based on a (0302222224) code that was proposed at one time for encoding the output of a 19-level quantizer in a 34 Mbit/s *YUV* codec. In the analysis that follows, the probability distribution of the symbols is taken to be that implied by the codeword lengths. Code 8, given in Table 5 and Fig. 7, is the result of applying

Table 5
Two codes for a 19-level DPCM quantizer

Symbol C_i	Probability P_i	Code 8	Code 9
0	0.25	00	00
+1	0.25	01	01
−1	0.25	10	10
+2	0.0625	1100	1100
−2	0.0625	1101	1101
+3	0.0313	11100	111000
−3	0.0313	11101	111001
+4	0.0156	111100	111010
−4	0.0156	111101	111100
+5	0.0078	1111100	111101
−5	0.0078	1111101	111110
+6	0.0039	11111100	11101100
−6	0.0039	11111101	11101101
+7	0.0019	111111100	11101110
−7	0.0019	111111101	11101111
+8	0.0010	1111111100	11111100
−8	0.0010	1111111101	11111101
+9	0.0010	1111111110	11111110
−9	0.0010	1111111111	11111111

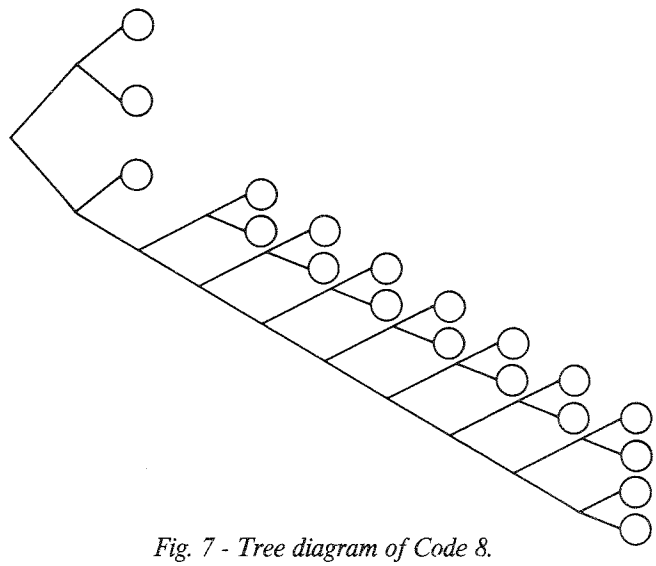


Fig. 7 - Tree diagram of Code 8.

Huffman's algorithm to this probability distribution. The bit rate of this code is 2.75 bits per symbol.

Code 8 appears to obey the conditions L and X discussed above as far as possible. We would therefore expect it to have a low value of Es. However, in an attempt to apply the principle introduced in Section 5.2, we compare Code 8 with another code which has a slightly different set of codeword lengths, (03020608). Such a code, like Code 7, has no codewords of odd length, and it must again be stressed that it would be a bad choice if bit-loss or bit-gain errors were possible. Code 9, given in Table 5 and Fig. 8, is the result of a heuristic design process that not only attempts to obey conditions L and X but also ensures that 00 is a synchronizing codeword, as defined in Section 2.3. Its bit rate is 2.77 bits per symbol, so this code is only slightly sub-optimal for the probability distribution given in Table 5.

Table 6 gives the results of calculating and estimating Es for Codes 8 and 9.

Table 6
Comparison of two 19-level DPCM codes

Code	Es	$P(S\|I)$	P_∞	P_0	Es_∞	Es_0
8	2.50	0.6222	0.2375	0.5132	2.59	1.74
9	1.31	0.7510	0.9942	0.8117	1.25	1.31

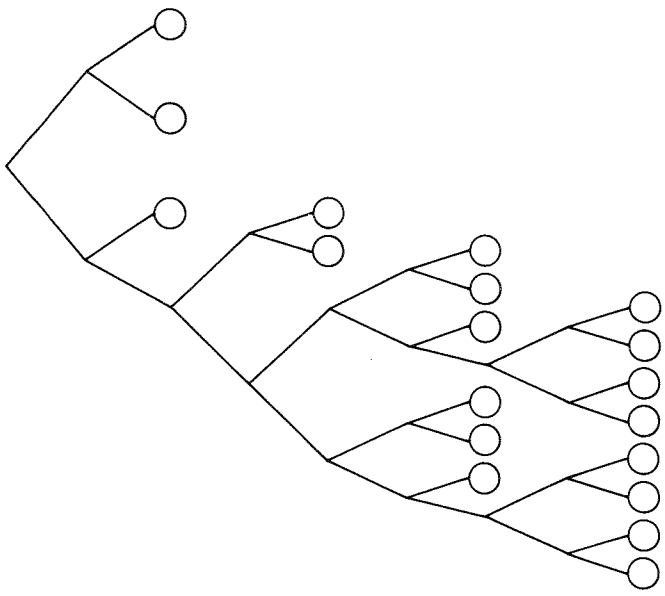


Fig. 8 - Tree diagram of Code 9.

These results show that an improvement of a factor of nearly two can be obtained if a very slightly sub-optimal code is used. They also indicate that the highly simplified method of calculating Es cannot always be trusted.

6. FURTHER WORK

In this Report no consideration has been given to trying to minimize the number of symbols lost or gained following an error. This is clearly an important aim in picture coding. It seems likely that codes with short expected resynchronization times will also perform well in this respect, although more weight may have to be given to maximizing $P(S|I)$, the probability of immediate resynchronization, on which the expected number of symbols lost or gained is likely to depend heavily. We have also seen that the improvements obtained in Codes 7 and 9 tend to increase $P(S|I)$.

Another aim in code design might be to allocate the symbols in such a way that errors have as little effect as possible on the subjective quality of the picture.

7. CONCLUSIONS

The Report has outlined a method of calculating the expected resynchronization time Es of a variable-length code, together with two simplified methods of calculating approximate estimates of Es. The use and validity of these methods have been illustrated by a number of examples. The results have indicated that, for a given set of codeword lengths, the value of Es can be made small by designing the code according to the following two principles, in order of priority:

(i) Short codewords should be suffixes of longer codewords;

(ii) Bit-reversal errors in codewords should produce other codewords.

In addition, we have illustrated that further improvements can be obtained by choosing a slightly sub-optimal code with a different set of codeword lengths. In particular, if the only errors are of the bit-reversal kind, codes that have no words of odd length seem to perform well.

8. ACKNOWLEDGMENT

The author is indebted to Dr. N.D. Wells who read the initial draft and made many valuable suggestions which have been embodied in the final version of this Report.

9. REFERENCES

1. KNEE, M.J. and WELLS, N.D. 1986. Comparison of Adaptive and Non-adaptive DPCM Systems. BBC Research Department Report to be published.

2. WELLS, N.D. 1987. Bit-rate Reduction for Long-Distance Transmission: An Introductory Review. BBC Research Department Report to be published.

3. HUFFMAN, D.A. 1951. A Method for the Construction of Minimum-redundancy Codes. Proc. IRE, Vol.40, September 1952, pp 1098-1101.

4. RUDNER, B. 1970. Construction of Minimum-redundancy Codes with an Optimum Synchronizing Property. IEEE Trans. on Information Theory, Vol. IT-17, No. 4, July 1971.

5. NEUMANN, P.G. 1962. Efficient Error-limiting Variable-length Codes. IRE Trans. on Information Theory, Vol. IT-8, No.4, July 1962, pp 292-304.

6. NEUMANN, P.G. 1970. Self-synchronizing Sequential Coding with Low Redundancy. Bell System Technical Journal, Vol.50, No.3, March 1971.

7. GHARAVI, H. 1985. Conditional Entropy Coding of Digital Pictures. J.I.E.R.E. Vol.56, No.5, May 1986, pp 213-218.

8. HATCHER, T.R. 1968. On a Family of Error-correcting and Synchronizable Codes. IEEE Trans. on Information Theory, Vol.15, No.5, September 1969, pp 620-624.

9. FERGUSON, T.J. and RABINOWITZ, J.H. 1983. Self-synchronizing Huffman Codes. IEEE Trans. on Information Theory, Vol. IT-30, No.4, July 1984, pp 687-693.

10. MAXTED, J.C. and ROBINSON, J.P. 1984. Error Recovery for Variable Length Codes. IEEE Trans. on Information Theory, Vol. IT-31, No. 6, November 1985, pp 794-801.

APPENDIX 1

Huffman's algorithm

Huffman's algorithm begins with a set of probabilities P_i and generates an optimal set of codewords C_i. The version of the algorithm used in the investigations described in the main text is as follows. (Note the significance of indenting in the description of the algorithm; instructions at the same level of indentation are to be considered together provided there are no instructions at a lower level between them.)

Initially, all n codewords C_i are empty (they will be built up as the algorithm progresses).

Form a list of n sets S_i of codewords with associated probabilities Q_i (Q_i will be the sum of the probabilities of all the codewords in the set S_i. Initially, each set S_i contains just one codeword C_i, and so $Q_i = P_i$.)

Repeat $n-1$ times

Find the smallest probability Q_j and the next-smallest probability Q_k in the list.

Append a **0** to the left of all codewords in set S_j

Append a **1** to the left of all codewords in set S_k

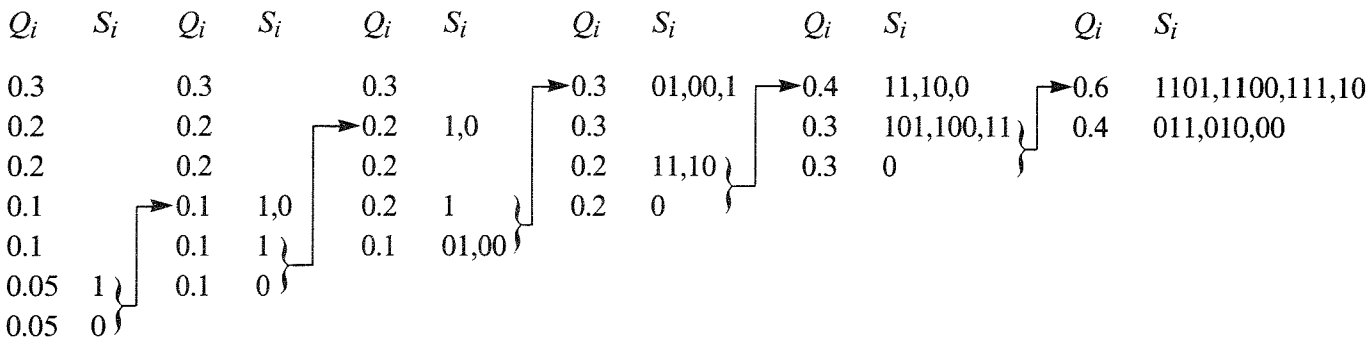
Let $S_j = S_j \cup S_k$ and $Q_j = Q_j + Q_k$ (i.e. combine sets S_j and S_k)

Remove S_k and Q_k from the list

End

The codewords can be assigned to the symbols either according to the path taken by each codeword as it is built up, or in lexicographical order within each codeword length. The latter method was used to generate Code 1.

Example: Generating Code 1



APPENDIX 2

Algorithm for reducing the signal flow graph

This Appendix describes how the expected resynchronization time Es is calculated, given the transition probabilities defined in Section 4. For convenience of implementing the algorithm, states I and S are thought of as one state T_0; then $P(T_0|\,T_k) = P(S|\,T_k)$ and $P(T_j|\,T_0) = P(T_j|\,I)$. In the description that follows, N_t is the number of error states and M is the maximum degree to which binomial expansions of polynomials are recorded. Indentations should be interpreted as in Appendix 1.

Set up a three-dimensional table of elements $F(m, j, k)$ with $m = 0,...,M$ and $j, k = 0,...,N_t$ where $F(m, j, k) =$ coefficient of z^m in the binomial expansion of the polynomial F_{jk} linking state k to state j; initially, $F(1, j, k) = P(T_j | T_k)$ and all other elements are zero.

Repeat for every error state T_r, $r = 1,...,Nt$:

Eliminate state T_r, as follows:

Calculate coefficients K_m of the binomial expansion of $(1 - F_{rr})^{-1}$, as follows:

For $m = 0,...,M$, set $K_m = 0$, $K'_m = 0$

Repeat for $m = 0,...,M$:

For $m' = 0,...,M$, set $K''_{m'} = K'_{m'}$, $K'_{m'} = 0$

Repeat for $m' = 0,...,M$

Repeat for $m'' = 0,...,M-m'$

Add $K''_{m'} \cdot F(m'', r, r)$ to $K'_{m'+m''}$ and to $K_{m'+m''}$

Repeat for every ordered pair of error states T_j and T_k that have not yet been eliminated:

Calculate new coefficients F_m of the binomial expansion of F_{jk}, as follows:

Repeat for $m = 1,...,M$

Repeat for $m' = 1,...,M$

Repeat for $m'' = 1,...,M$

If $m + m' + m'' < M$, add $F(m, r, k) \cdot F(m', j, r) \cdot K_{m''}$ to $F(m+m'+m'', j, k)$

When all the error states have been eliminated, F_{00} is the polynomial linking state I to state S, so

$$Es = \sum_{m=1}^{M} m \cdot F(m,0,0)$$

End

As it stands, a large value of M is needed in general to give an accurate value of Es, particularly if any of the $P(T_j | T_j)$ is near 1. This makes the algorithm rather slow. However, it has been observed that, for large m, the coefficients of the final polynomial F_{00} decay exponentially. Hence, a smaller value of M can be used and the effect of the remaining terms on Es can be calculated. The correction to be added to Es is:

$$-F(M,0,0) \cdot \frac{\sigma}{\log_e \sigma} \cdot \left(M + 1 - \frac{1}{\log_e \sigma}\right)$$

where

$$\sigma = \frac{F(M,0,0)}{F(M-1,0,0)}$$

With this correction, $M = 10$ was large enough for an accurate estimate of Es for a selection of larger codes for which a comparison (without a correction) was made with $M = 100$.